

CS3500 HW2 Solutions

1.23 (5 points each)

a. Let L be the language and assume L is regular. Let p be the pumping length and s be the string 0^p10^p . Because s is a member of L and s has length more than p , the pumping lemma guarantees that s can be split into three pieces, $s = xyz$ satisfying the three conditions of the pumping lemma. Because of condition 3 ($|xy| \leq p$), y can only consist of 0s. Then the string xy^2z will not be in L since the number of 0s before 1 will exceed the number of 0s after 1.

b. Let L be the language and assume it is regular. Then its complement $\bar{L} = \{0^n1^n | n \geq 0\}$ is also regular since the class of regular language is closed under complement. But we know in class that \bar{L} is not regular and thus we have a contradiction.

c. Let L be the language and assume it is regular. Then its complement \bar{L} is also regular. Furthermore, the language $\bar{L} \cap \{0^*1^*\}$ is also regular since the class of regular language is closed under intersection. We know that $\bar{L} \cap \{0^*1^*\} = \{0^n1^n | n \geq 0\}$ is not a regular language and this is a contradiction.

Note: The complement of L is not the language in part b). \bar{L} should be a UNION of the strings NOT of the form $\{0^*1^*\}$ and the language in part b).

d. Let L be the language and assume it is regular. Let p be the pumping length and s be the string 0^p10^p . Following the same reasoning in part a), we can show that either we pump up or pump down, the resulting string will not be a palindrome.

2.4 (5 points each)

a.

$S \rightarrow A1A1A1A$

$A \rightarrow A0 | A1 | \epsilon$

b.

$S \rightarrow 0A0 | 1A1 | 0 | 1 | \epsilon$

$A \rightarrow A0 | A1 | \epsilon$

c.

$S \rightarrow 1A | 0A$

$A \rightarrow 1S | 0S | \epsilon$

d.

$S \rightarrow 0S0 | 0S1 | 1S0 | 1S1 | 0$

e.

$S \rightarrow SAR | RAR$

$A \rightarrow 1A | 1$

$R \rightarrow ROR1 | R1R0 | \epsilon$

f.

$S \rightarrow 0S0 | 1S1 | 0 | 1 | \epsilon$

2.7 (5 points each)

a. We'll use $\Gamma = \{a, b, \bar{a}\}$, where \bar{a} represents a deficit of a 's. Read symbols from the input. If an a is read and there is \bar{a} on the stack, pop \bar{a} . If an a is read and there is not \bar{a} on the stack, push a . If a b is read, repeat the following exactly twice: if a is on the stack, pop a ; otherwise push \bar{a} . If we run out of input when the stack is empty, accept. Otherwise reject.

d. There are two cases we should consider: $i = j$ or $i \neq j$.

Case 1: Nondeterministically decide $i \neq j$ and do the following:

a. Repeat:

If there is no more input, reject. Otherwise, nondeterministically decide if this is string i . If not, read input through to the next $\#$ or end of input. If so, read input through to the next $\#$ or end of input but push every symbol read onto the stack, except the $\#$, and break out of this loop.

b. Repeat:

If there is no more input, reject. Otherwise, nondeterministically decide if this is string j . If not, read input through to the next $\#$ or end of input. If so, read input through to the next $\#$ or end of input but compare each symbol with the symbol on the stack, popping as we go. If they match and the stack is empty and we've reached $\#$ or end of input, read the rest of the input and accept. Otherwise, reject.

Case 2: Nondeterministically decide $i = j$ and do the following:

Repeat:

If there is no more input, reject. Otherwise, nondeterministically decide if this is string i .

a. If not, read input through to the $\#$ or end of input.

b. If so, read input symbols and push them onto the stack. If $\#$ is read or end of input is reached, reject. Nondeterministically decide when we hit the middle of the string. Nondeterministically read a , b , or nothing. Now read the rest of the string (up to $\#$ or end) and compare to the stack, popping as we go. If they match and the stack is empty and we've read the whole string, accept. Otherwise reject.

2.17 a. (15 points) Let M_C be a PDA accepting C and M_R be a DFA accepting R . We construct a PDA M accepting $C \cap R$ in a manner similar to Theorem 1.12 in Sipser. Say $M_C = (Q_C, \Sigma, \Gamma, \delta_C, q_C, F_C)$ and $M_R = (Q_R, \Sigma, \delta_R, q_R, F_R)$. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ with the following: $Q = Q_C \times Q_R$, $q_0 = (q_C, q_R)$, $F = \{(r_1, r_2) \mid r_1 \in F_C \text{ and } r_2 \in F_R\}$. Define δ like this: Say for some $r_1 \in Q_C$, $a \in \Sigma$, and $t \in \Gamma$, $\delta_C(r_1, a, t) = \{(q_1, t_1), \dots, (q_n, t_n)\}$ for some $n \geq 0$. Then we define $\delta((r_1, r_2), a, t) = \{((q_1, \delta_R(r_2, a)), t_1), \dots, ((q_n, \delta_R(r_2, a)), t_n)\}$. Basically M simulates M_C and M_R together. M_R doesn't pay attention to M 's stack, so M is a valid PDA with one stack, which it gets from M_C . A string accepted by both M_C and M_R will end up in an accept state in each, and so it will follow

these paths in M as well and be accepted by M . If a string is rejected by either, it won't get to an accept state of M .

b. (5 points) Let $B = a^*b^*c^*$ and $C = \{a^ib^ic^i \mid i \geq 0\}$. Note that $A \cap B = C$. Suppose A is context-free. Since B is regular, by part **a** above we have that C is context-free. But this contradicts Example 2.20 in Sipser, so A cannot be context free.

2.18 (5 points each)

a. Assume that this language L is context-free and it has a pumping length p . Consider the string $s = 0^p1^p0^p1^p$. Since s is in the language and the length of s is greater than p , s can be divided into $uvxyz$ satisfying all three conditions. There are two possibilities: each of v and y are made up entirely of one symbol, or one of them contains both 0 and 1 somewhere in it. If the latter is true, pumping will clearly give 0s and 1s in the wrong order. If the former is true, at most two of the sets of symbols will increase when v and y are pumped, but we need all four sets to increase to stay in the language. Thus no division is possible, and so our assumption that L is context free is contradicted.

b. Consider the string $s = 0^p\#0^{2p}\#0^{3p}$. There are two possibilities: each of v and y are made up of only 0s, or one of them contains a $\#$. In the latter case, pumping will clearly give too many $\#$. In the former case, only two of the sets of zeros could increase when v and y are pumped, and we need all three to increase to stay in the language. Thus we have reached a contradiction.

c. Consider the string $s = a^pb^p\#a^pb^p$. First, if v and y are both in the first part of the string, pumping makes this string too long to be a substring of the second part. Likewise, if v and y are both in the second part of the string, pumping down (that is, $i = 0$ in the pumping lemma) makes this string too short to be a superstring of the first part. (The same occurs if either v or y is empty.) Now, condition 3 says that s can be divided so that $|vxy| \leq p$. Therefore, since we have eliminated the other possibilities (and clearly neither v nor y can contain $\#$), we conclude that v must consist of bs from the first part of the string and y must consist of as from the second part of the string. But now when they are pumped, the first part cannot be a substring of the second part, because it has too many bs. Thus no division of s that meets all the conditions exists, and we have a contradiction.

d. Consider the string $s = a^pb^p\#a^pb^p$. Note that s is in the language ($k = 2$ for s) and it has length greater than p , so the pumping lemma applies. If v and y are both in the same part of the string, then pumping will not keep the string in the language, because the two parts will no longer match. If they are each in one of the parts of the string and do not contain $\#$, the argument above in part **c** applies again and the strings cannot match when pumped. Now, for this language duplicating the $\#$ is not an immediate problem. However, if one of v

or y contains the $\#$ in s , pumping down will eliminate it, and the string will no longer be in the language because every string in the language contains at least one $\#$ symbol. This covers all possibilities, so we have a contradiction.